

Apache Spark Word Count App

User

2025-01-13, 01:15:56

Generate a Apache Spark application using Scala language to solve the problem! Given a plain text UTF-8 file URL and a output CSV file URL, write in the output file the occurrence sum of each word in the input file.

Assistant Gemini 2.0 Flash Experimental

2025-01-13, 01:15:56

```
```scala import org.apache.spark.sql.SparkSession object WordCount { def main(args:
Array[String]): Unit = { if (args.length != 2) { System.err.println("Usage: WordCount ")
System.exit(1) } val inputFile = args(0) val outputFile = args(1) val spark = SparkSession
.builder() .appName("Word Count") .getOrCreate() import spark.implicits._ // Read the text file
into a Dataset of Strings val textFile = spark.read.text(inputFile).as[String] // Split each line into
words and flatten the Dataset val words = textFile.flatMap(_.split(" ")) // Group by word and
count occurrences val wordCounts = words.groupBy("value").count() // Write the result to a
CSV file wordCounts.coalesce(1).write .option("header", "true") .csv(outputFile) spark.stop() } }
```
```

****Explanation:**** 1. ****Import SparkSession:**** This line imports the necessary class to create a Spark session. 2. ****Main Function:**** The `main` function is the entry point of the application. It takes an array of strings as arguments. 3. ****Argument Check:**** Checks if the correct number of arguments is provided (input and output file paths). 4. ****Create SparkSession:**** It creates a SparkSession, the entry point to Spark functionality, using the builder pattern. The `appName` is set to "Word Count". 5. ****Import implicits:**** This line imports the implicit conversions, which enables us to use methods like `toDF` or `as`. 6. ****Read Text File:**** It reads the text file from the given input path into a Dataset of Strings. 7. ****Split into Words:**** The lines are transformed into words by splitting them on spaces using `flatMap`. This results in a Dataset of individual words. 8. ****Group and Count:**** The `groupBy("value")` groups identical words together, and the `count()` aggregates the number of occurrences for each word. 9. ****Write to CSV:**** The resulting `wordCounts` Dataset is saved as a CSV file to the given output path, with the header included in the CSV file. The `coalesce(1)` operation merges all the data into a single partition before writing, so the output will be in a single file. 10. ****Stop SparkSession:**** The `spark.stop()` command shuts down the Spark session. ****How to use:**** 1. Save the code in a file named `WordCount.scala`. 2. Compile the code using `scalac WordCount.scala`. 3. Package the compiled code into a JAR file. 4. Run the application using `spark-submit --class WordCount`. Make sure to replace ``, `,`, and `` with the actual paths.